

# Pontifica Universidade Católica do Rio Grande do Sul

## Ciências da Computação

### Algoritmos e Estruturas de Dados 1 – Trabalho 3.

Prof. Edson Ifarraguirre Moreno

Giovani Schardong

Mateus F. Poletto

O último trabalho proposto pela cadeira de Algoritmos e Estrutura de Dados 1 consistia em analisar o entendimento dos estudantes a respeito do funcionamento de algoritmos de árvores (Tree), podendo estas serem binárias, básicas, entre outras. Foi proposto para nós desenvolvermos um código que facilitaria o entendimento de pergaminhos Vikings que continham guerreiros Vikings com a quantidade de terras dos meus e seus filhos com suas respectivas quantidades de terras.

Na explicação da leitura do pergaminho, foi repassado para nós que a primeira linha sempre seria a quantidade de terras do primeiro ancestral da linhagem viking. Nas linhas seguintes teríamos o formato “(Nome do pai) (Nome do filho) (Quantidade de terras do filho)”, sendo que o pai que aparecer na segunda linha do pergaminho seria o dono das terras da primeira linha. As terras que aparecem no final da linha nunca são o total de terras do filho no final de sua vida, pois além dessas terras, devemos somar o valor da terra que seria passado de herança pelo seu pai. A herança representa a quantidade de terras divididas pelos seus filhos.

Nossa missão era encontrar qual era o último descendente vivo que possuía mais terras. Portanto segue a demonstração do algoritmo elaborado para descobrir o grande felizardo da linhagem Viking.

#### **Leitura do arquivo**

Os arquivos enviados para teste no programa são enviados em formato txt respeitando o padrão existente no pergaminho apenas possuindo números e os nomes dos Viking, sendo cada informação separada por espaços, cada linha é formada pelo nome do pai “espaço” nome do filho “espaço” quantidade de terras do filho, em exceção da primeira linha que apenas possui a quantidade de terras do primeiro ancestral.

Exemplo de arquivo texto:

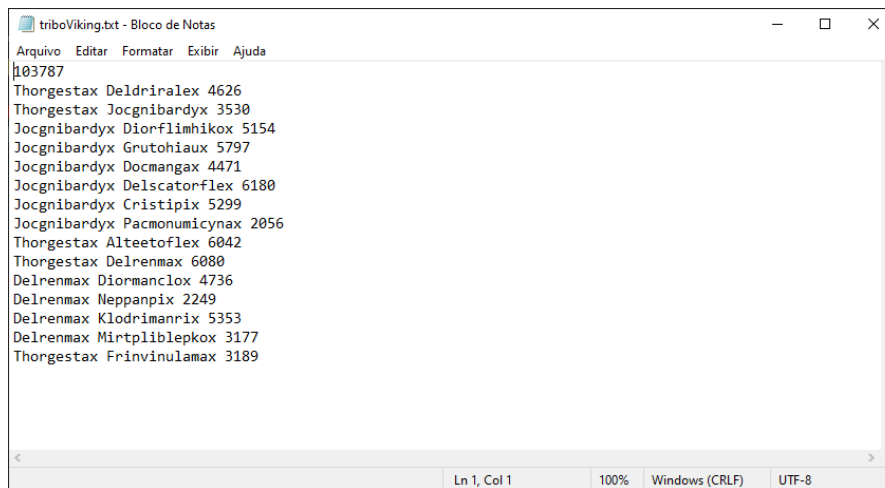


Imagem 1

Após essa verificação se o arquivo está nos padrões do pergaminho, o programa é iniciado na classe App.

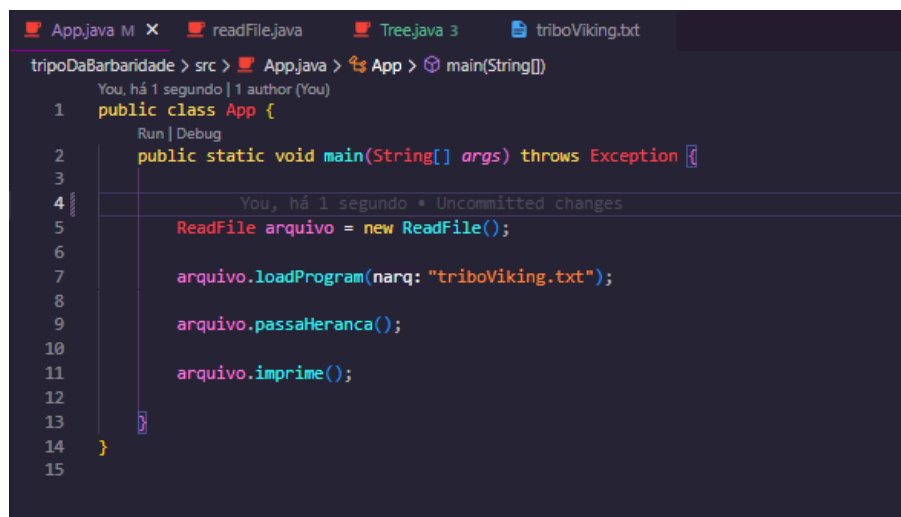
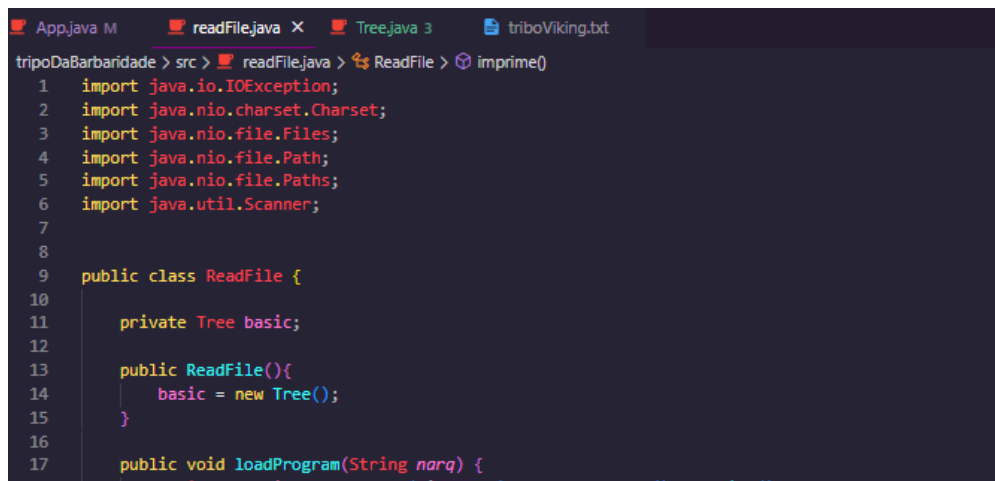


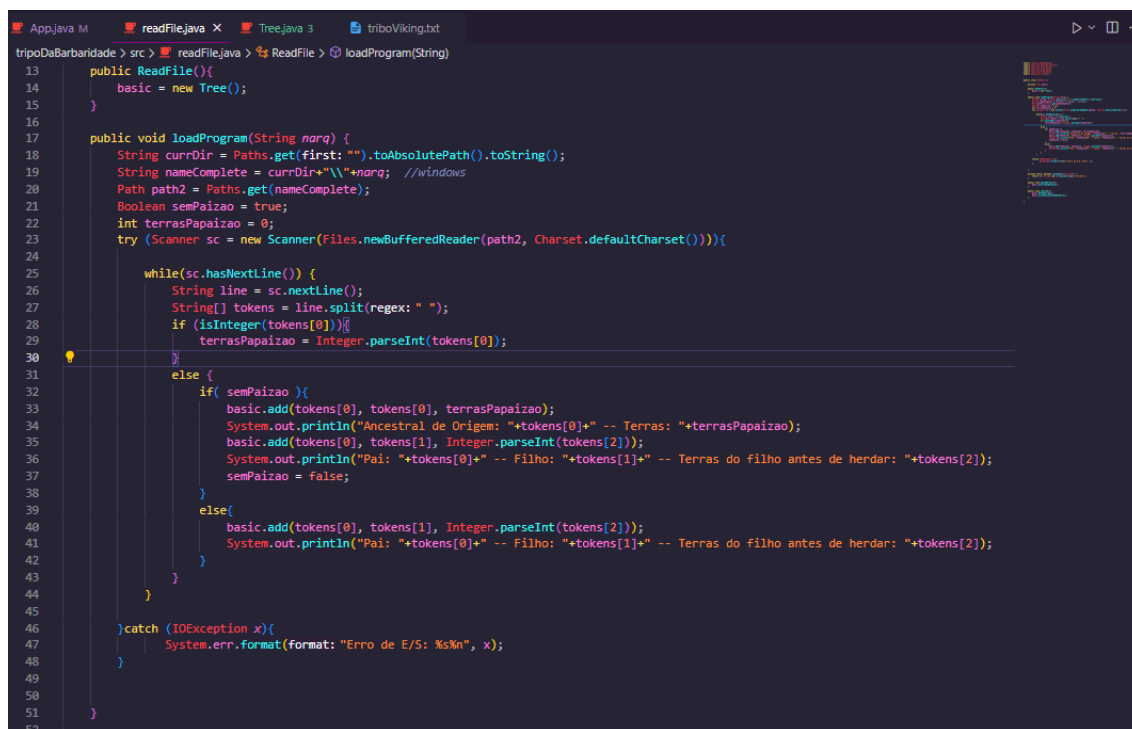
Imagem 2

É iniciado o programa carregando uma instância da classe ReadFile, quando carregada essa instância o método construtor da mesma carregará uma instância da classe Tree para a variável **basic** que será o objeto que trabalharemos o pergaminho. Então nesta instância de Readfile passamos o nome do arquivo com sua extensão como parâmetro para o método loadProgram que fará a leitura do pergaminho.



```
App.java M readFile.java X Tree.java 3 triboViking.txt
tripoDaBarbaridade > src > readFile.java > ReadFile > imprime()
1 import java.io.IOException;
2 import java.nio.charset.Charset;
3 import java.nio.file.Files;
4 import java.nio.file.Path;
5 import java.nio.file.Paths;
6 import java.util.Scanner;
7
8
9 public class ReadFile {
10
11     private Tree basic;
12
13     public ReadFile(){
14         basic = new Tree();
15     }
16
17     public void loadProgram(String narg) {
```

Imagem 3



```
App.java M readFile.java X Tree.java 3 triboViking.txt
tripoDaBarbaridade > src > readFile.java > ReadFile > loadProgram(String)
13 public ReadFile(){
14     basic = new Tree();
15 }
16
17 public void loadProgram(String narg) {
18     String currDir = Paths.get(first: "").toAbsolutePath().toString();
19     String nameComplete = currDir+"\\ "+narg; //windows
20     Path path2 = Paths.get(nameComplete);
21     Boolean semPapaizao = true;
22     int terrasPapaizao = 0;
23     try (Scanner sc = new Scanner(Files.newBufferedReader(path2, Charset.defaultCharset()))){
24
25         while(sc.hasNextLine()) {
26             String line = sc.nextLine();
27             String[] tokens = line.split(regex: " ");
28             if (isInteger(tokens[0])){
29                 terrasPapaizao = Integer.parseInt(tokens[0]);
30             }
31             else {
32                 if( semPapaizao){
33                     basic.add(tokens[0], tokens[0], terrasPapaizao);
34                     System.out.println("Ancestral de Origem: "+tokens[0]+" -- Terras: "+terrasPapaizao);
35                     basic.add(tokens[0], tokens[1], Integer.parseInt(tokens[2]));
36                     System.out.println("Pai: "+tokens[0]+" -- Filho: "+tokens[1]+" -- Terras do filho antes de herdar: "+tokens[2]);
37                     semPapaizao = false;
38                 }
39                 else{
40                     basic.add(tokens[0], tokens[1], Integer.parseInt(tokens[2]));
41                     System.out.println("Pai: "+tokens[0]+" -- Filho: "+tokens[1]+" -- Terras do filho antes de herdar: "+tokens[2]);
42                 }
43             }
44         }
45     }catch (IOException x){
46         System.err.format(format: "Erro de E/S: %s\n", x);
47     }
48 }
49
50
51
52 }
```

Imagem 4

A classe loadProgram recebe o parâmetro que enviamos que é referente ao nome do arquivo, é armazenado em formato String o endereçamento raiz na pasta que estamos trabalhando na variável currDir. Então na variável nameComplete criamos o endereçamento em formato String somando os valores de currDir, “\\”(caso Windows) e o nome do arquivo que foi recebido de parâmetro no método. Então transformamos esse endereçamento para o formato Path na variável path2 para garantir a leitura do arquivo. Então começamos a realizar a leitura do arquivo lendo linha a linha do pergaminho, a leitura só acaba quando não houver mais linhas para ler.

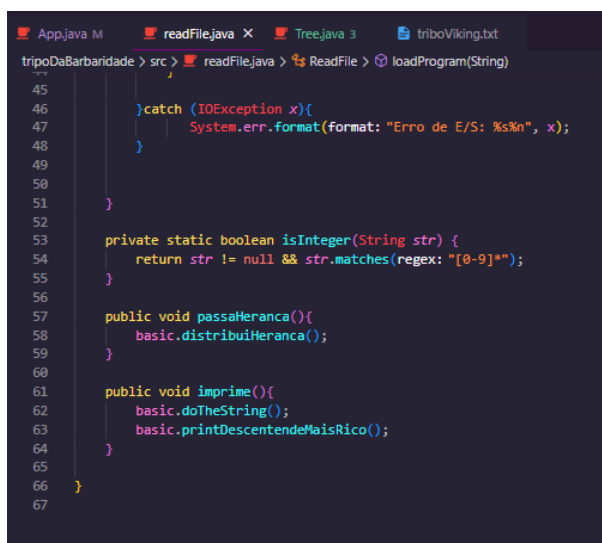
Em todas as passadas do while, é criada line que armazena a linha inteira que estamos trabalhando, após isso em tokens criados um array que é formado pelos itens

que formam a linha, sendo cada item dividido por um espaço. Então testamos para ver se o primeiro item da linha não representa um número, pois se o mesmo representa um número devemos salvá-lo para utilizarmos depois quando tivermos o nome do primeiro ancestral da linhagem.

Na próxima passada o processo é o mesmo até encontrarmos novamente a primeira condição, como já nas próximas linhas nunca teremos um número como primeiro item, não iremos entrar nessa condição e partiremos sempre para o “else”.

No “else” encontraremos a última condição e teste para a criação da nossa árvore da linhagem viking em que se estamos na segunda linha, ainda não teremos um ancestral comum cadastrado, então entraremos nesse if e na primeira adição passaremos a posição no array em tokens do nome do pai e sua quantidade de terras. Em seguida adicionaremos o primeiro filho dele, novamente chamamos o método add do objeto basic e passamos o nome do pai, do filho e quantidade de terras do filho em suas respectivas posições no array de tokens. E então atribuímos a variável “semPaizao” o valor de false pois já possuímos o primeiro ancestral e não precisaremos mais entrar nessa condição.

As próximas linhas serão todas iguais, todas sendo direcionadas para o “else” dentro do “else” em que o processo de adição do filho é o mesmo do já descrito anteriormente pelo método add. Para economizar processamento, a primeira impressão no terminal é feita na ocorrência de leitura do pergaminho.



```
App.java M readFile.java x Tree.java 3 triboViking.txt
triboDaBarbaridade > src > readFile.java > ReadFile > loadProgram(String)
45
46     }catch (IOException x){
47         System.err.format(format: "Erro de E/S: %s\n", x);
48     }
49
50
51 }
52
53 private static boolean isInteger(String str) {
54     return str != null && str.matches(regex: "[0-9]*");
55 }
56
57 public void passaHeranca(){
58     basic.distribuiHeranca();
59 }
60
61 public void imprime(){
62     basic.doTheString();
63     basic.printDescendenteMaisRico();
64 }
65
66 }
67
```

### Dividindo a Herança:

A próxima etapa do nosso projeto é dividir a herança dos pais e passar para seus descendentes. Por isso chamamos o método passaHeranca do objeto arquivo (Imagem 2) que iniciará o método distribuiHeranca do objeto basic.

Para a classe Tree foi utilizado como base a implementação desenvolvida em aula. Foram realizadas as seguintes alterações:

```

public class Tree {
    //NODO BASICO PARA A CONSTRUÇÃO DA ÁRVORE
    private class TreeNode{
        public String nome;
        public TreeNode father;
        private TreeNode [] children;
        private int nChild;
        public int terras;
        public int nLinhagem;
        public String espacos = "";
        public int heranca;

        public TreeNode (String guerreiro, Integer element){
            father=null;
            children=new TreeNode[10];
            nome=guerreiro;
            terras=element;
            nChild=0;
        }
        public void addSubtree (TreeNode n){
            if(nChild==children.length)
                grow();
            children[nChild] = n;
            n.father=this;
            nChild++;
            n.nLinhagem = this.nLinhagem + 1;
            n.espacos = this.espacos + " ";
        }
    }
}

```

Na Criação e construção da árvore, foram criadas as variáveis: int terras, que armazena as terras do nodo ou da folha. Int nLinhagem, que armazena qual é a geração do elemento da árvore. String espacos, que foi utilizada para a legibilidade da impressão da árvore. Int heranca, que armazena quantas terras os filhos de um pai devem receber.

Na criação de um TreeNode, que recebe como parâmetro, o nome do guerreiro e as suas terras, Father é inicializado como null, children recebe um novo treenode de tamanho 10, "nome", recebe o nome do guerreiro, "terras" recebe as terras desse guerreiro, e nChild é inicializado como 0.

O método addSubtree, que recebe como parâmetro o TreeNode, analisa se nChild é igual a o tamanho de children, se for, ele chamao método Grow. Children na posição nChild recebe o Parâmetro treeNode. A variável nLinhagem é incrementada em 1, e a variável espacos é incrementada com um " ".

```

public void passaHeranca(){
    heranca = terras/nChild;
    for (int i= 0; i < nChild; i++){
        children[i].terras += heranca;
    }
    terras = 0;
}

```

O método `passaHeranca`, não recebe parâmetro, ele atribui a variável `heranca` a divisão da variável `terras` por `nChild`, e enquanto a variável `i` for menor que o valor de `nChild`, para cada posição de `children`, suas terras são incrementadas com o valor da variável `heranca`. A variável `terras` recebe valor 0.

Foi Criada a variável privada do tipo `TreeNode` `descendenteMaisRico`.

Na classe `searchNode`, que recebe `String value` e `TreeNode ref` como parâmetro, ele analisa se `ref` não é nulo, se não for, ele analisa se `ref` ao nome do guerreiro é igual ao parâmetro `value`, se for, ele retorna `ref`.

```

private TreeNode root;
private int nElements;
private TreeNode descendenteMaisRico;

public Tree(){
    this.root=null;
    this.nElements=0;
}

// método privado elaborado na versão 0.2
private TreeNode searchNode(String value, TreeNode ref){

    if(ref!=null){
        if(ref.nome.equals(value))
            return ref;
    }
}

```

O método `add`, recebe os parâmetros `Father`, `son` e `terras`, se `nElements` for igual a 0 `this.root` recebe um novo `TreeNode` com `son` e `terras` como parâmetros. A variável `descendenteMaisRico` recebe a variável `root`. E a linhagem dessa `root` é definida como 1.

Se for diferente de 0, a variável `aux` é atribuída a chamada do método `searchNode`, passando os parâmetros `Father` e `root`. Ele analisa se `aux` é igual a `null`, se for, retorna `false`, se não for `aux` chama `addSubtree` com um novo `TreeNode` com os parâmetros `son` e `terras`.

O `nElements` é acrescido em 1 e o método retorna `true`.

O método `distribuiHeranca` chama o método `herancaDoPapai` com o parâmetro `root`.

```

public boolean add(String father, String son, Integer terras){
    TreeNode aux;
    if(nElements==0){
        this.root=new TreeNode(son, terras);
        descendenteMaisRico = root;
        root.nLinhagem = 1;
    }
    else{
        aux = searchNode(father, root);
        if(aux==null)
            return false;
        else
            aux.addSubtree(new TreeNode(son, terras));
    }
    nElements++;

    return true;
}

public void distribuiHeranca(){

    herancaDoPapai(root);

}

```

O método boolean herancaDoPapai, recebe como parâmetro o TreeNode “VeioDaLancha”. Ele analisa se VeioDaLancha chamando o método getSubTreeSize é maior que 0, se for, enquanto a variável i for menor que VeioDaLancha chamando o método getSubTreeSize, ele chama o método passaHeranca e analisa se VeioDaLancha chamando o método getSubtree com parâmetro i, chamando o método getSubTreeSize, é maior que 0, se for, ele chama o método herançaDoPapai com o parâmetro VeioDaLancha.getSubtree com parâmetro i.

Ao Final retornando true.

```

public boolean herancaDoPapai(TreeNode VeioDaLancha){

    if (VeioDaLancha.getSubtreeSize() > 0){
        for (int i =0; i < VeioDaLancha.getSubtreeSize(); i++){
            VeioDaLancha.passaHenanca();
            if (VeioDaLancha.getSubtree(i).getSubtreeSize() > 0)
                herancaDoPapai(VeioDaLancha.getSubtree(i));
        }
    }
    return true;
}

```

O método doTheString chama o método printValueResult passando o parâmetro root duas vezes.

O método printValueResult irá imprimir a árvore, organizando a qual pai cada filho se refere, e se os filhos possuem filhos ou não.

```
public void doTheString(){  
    printValueResult(root, root);  
}  
  
private void printValueResult(TreeNode ref, TreeNode refEspacos){  
    if(ref!=null){  
        if (ref.nome.equals(refEspacos.nome))  
            System.out.print(ref.nome+" Filhos: ");  
        else {  
            System.out.print(ref.espacos+ref.nome+" Filhos: ");  
        }  
        System.out.print(s: " [ ");  
        for(int i=0; i<ref.getSubtreeSize(); i++){  
            if (ref.getSubtreeSize() == 0)  
                System.out.print(s: " ] ");  
            printValueResult(ref.getSubtree(i), ref);  
        }  
    }  
}
```



O método publico printDescendenteMaisRico chama o método privativo printDescendenteMaisRico passando a variável root. Em seguida imprime o nome do descendenteMaisRico e as terras do descendenteMaisRico.

O método privativo printDescendenteMaisRico recebe a ref a root. O método compara o nLinhagem de cada um dos nodos a partir da raiz. Ele descobre quais folhas estão na maior linhagem, após isso ele analisa a quantidade de terras de cada um deles, o que possuir mais terras é definido como o Descendente mais rico.

```
public void printDescendenteMaisRico(){
    printDescendenteMaisRico(root);
    System.out.println();
    System.out.println("Descendente mais rico: "+descendenteMaisRico.nome+ " - Quantidade de Terras: "+descendenteMaisRico.terras);
}

private void printDescendenteMaisRico(TreeNode ref){
    if (ref != null){
        if (ref.getSubtreeSize() == 0){
            if (ref.nLinhagem > descendenteMaisRico.nLinhagem){
                descendenteMaisRico = ref;
            }
            else if (ref.nLinhagem == descendenteMaisRico.nLinhagem){
                if (ref.terras >= descendenteMaisRico.terras){
                    descendenteMaisRico = ref;
                }
            }
        }
        for (int i = 0; i < ref.getSubtreeSize(); i++){
            printDescendenteMaisRico(ref.getSubtree(i));
        }
    }
}
```

App:

```
public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {

        ReadFile arquivo = new ReadFile();

        arquivo.loadProgram(narq: "TriboEDSON.txt");
        // arquivo.loadProgram("TriboTESTE1.txt");
        // arquivo.loadProgram("TriboTESTE2.txt");
        // arquivo.loadProgram("TriboTESTE3.txt");
        //arquivo.loadProgram("TriboTESTE4.txt");

        arquivo.passaHeranca();

        arquivo.imprime();
    }
}
```

A classe App é a main. A cria um novo objeto do tipo ReadFile, chamado arquivo. A variável arquivo chama o método loadProgram e informa qual dos arquivos txt deve ser lido, logo após arquivo chama o método passaHeranca e após arquivo chama o método imprime.

## Execução do Programa

(exemplo de execução do programa)

```
Ancestral de Origem: Thorgestax -- Terras: 183787
Pai: Thorgestax -- Filho: Deldeiralex -- Terras do filho antes de herdar: 4626
Pai: Thorgestax -- Filho: Jcognibardyx -- Terras do filho antes de herdar: 3538
Pai: Jcognibardyx -- Filho: Diorflinhikox -- Terras do filho antes de herdar: 5154
Pai: Jcognibardyx -- Filho: Grutohiaux -- Terras do filho antes de herdar: 5797
Pai: Jcognibardyx -- Filho: Docmangax -- Terras do filho antes de herdar: 4471
Pai: Jcognibardyx -- Filho: Delscatorflex -- Terras do filho antes de herdar: 6188
Pai: Jcognibardyx -- Filho: Cristipix -- Terras do filho antes de herdar: 5229
Pai: Jcognibardyx -- Filho: Pacmonumicynax -- Terras do filho antes de herdar: 2856
Pai: Thorgestax -- Filho: Delrenmax -- Terras do filho antes de herdar: 6888
Pai: Delrenmax -- Filho: Diormanclox -- Terras do filho antes de herdar: 4736
Pai: Delrenmax -- Filho: Neppanpix -- Terras do filho antes de herdar: 2249
Pai: Delrenmax -- Filho: Klodrimanrix -- Terras do filho antes de herdar: 5353
Pai: Delrenmax -- Filho: Mirtpliblekox -- Terras do filho antes de herdar: 3177
Pai: Thorgestax -- Filho: Frinvinulamax -- Terras do filho antes de herdar: 3189
Thorgestax Filhos: [ Deldeiralex Filhos: [ Jcognibardyx Filhos: [ Diorflinhikox Filhos: [ Grutohiaux Filhos: [ Docmangax Filhos: [ Delscatorflex Filhos: [ Cristipix Filhos: [ Pacmonumicynax Filhos: [ Alteetoflex Filhos: [ Delrenmax Filhos: [ Diormanclox Filhos: [ Neppanpix Filhos: [ Klodrimanrix Filhos: [ Mirtpliblekox Filhos: [ Frinvinulamax Filhos: [
Descendente mais rico: Klodrimanrix - Quantidade de Terras: 12862
PS C:\Users\giovanni\Desktop\Barbara\Alesti-Basic\Free-Tribos>
```